NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION
OF A DATA MODEL
FOR THE NPS ARGOS PROJECT

by

Stefan A. H. Westman

March 1992

Thesis Advisor: Dr. Thomas Wu
Co advisor CDR B.B. Giannotti

Approved for public release: distribution is unlimited.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School	6b. OFFICE SYMBOL (*if applicable*): CS
6c. ADDRESS (*City, State, and ZIP Code*): Monterey, CA 93943-5000	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (*if applicable*)
8c. ADDRESS (*City, State, and ZIP Code*)	7b. ADDRESS (*City, State, and ZIP Code*): Monterey, CA 93943-5000
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION N

11. TITLE (*Include Security Classification*)
DESIGN AND IMPLEMENTATION OF A DATA MODEL FOR THE NPS ARGOS PROJECT

12. PERSONAL AUTHOR(S)
Stefan A. H. Westman

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Master's Thesis	FROM 04/90 TO 03/92	1992, March	76

16. SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the author and do not reflect the offici policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Logistic, US Navy, Relational database, RDBMS, Cals, SNAP, Oracle, Hypercard, Macintosh.

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The ARGOS project is a design and an implementation of a prototype multimedia database system developed as both Battle Group Commander's assessment tool and a shipboard data management tool. The original prototype was developed usir the HyperCard/Macintosh technology, taking advantage of its object-oriented properties and its user-friendly graphic interface. The major problem with the current implementation is that all information is located in a relatively slow and ineffecie database. Updates of information have to be hard-coded and access to other databases in or outside the current workir environment is not supported.

This thesis proposes an enhanced system taking advantage of relational database management technique. The propose system is based on the idea that all information, including images, button locations, and other system variables, shall be accesse from the relational database management system. This approach makes it possible to separate the user interface from the store data thus providing a platform independent environment. The enhanced system is developed using Oracle as the relation database management system. The user interface is built in Hypercard on the Macintosh. All data retrieval is based on ANS SQL.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Thomas Wu	22b. TELEPHONE (*Include Area Code*): (408) 646-2174	22c. OFFICE SYMBOL: CS/Wq

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE: UNCLASSIFIED

Approved for public release; distribution is unlimited

DESIGN AND IMPLEMENTATION OF A DATA MODEL FOR THE NPS ARGOS PROJECT

by

Stefan A. H. Westman
Major, Swedish Army
MS Civil Engineering, Swedish Defence Staff and War Collage, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

ABSTRACT

The ARGOS project is a design and an implementation of a prototype multimedia database system developed as both a Battle Group Commander's assessment tool and a shipboard data management tool. The original prototype was developed using the HyperCard/Macintosh technology, taking advantage of its object-oriented properties and its user-friendly graphical interface. The major problem with the current implementation is that all information is located in a relatively slow and inefficient database. Updates of information have to be hard-coded and access to other databases in or outside the current working environment is not supported.

This thesis proposes an enhanced system taking advantage of relational database management technique. The proposed system is based on the idea that all information, including images, button locations, and other system variables, shall be accessed from the relational database management system. This approach makes it possible to separate the user interface from the stored data thus providing a platform independent environment. The enhanced system is developed using Oracle as the relational database management system. The user interface is built in Hypercard on the Macintosh. All data retrieval is based on ANSI SQL.

TABLE OF CONTENTS

I. THESIS OBJECTIVES AND OUTLINE

This chapter briefly describes previous related work made in the "Paperless Ship" concept at Naval Postgraduate School. It also gives the objectives and the outline of the thesis.

A. PREVIOUS WORK

1. The ARGOS project

The ARGOS project [GIAN 89] is a design and an implementation of a prototype multimedia database system developed as both a Battle Group Commander's assessment tool and a shipboard data management tool. It is part of the feasibility exploration of the "Paperless Ship" concept. The original prototype was developed using the HyperCard/ Macintosh technology, taking advantage of its object-oriented properties and its user-friendly graphical interface. The ARGOS multimedia database is capable of handling images (graphics), signals (sound) and text.

The major problem with the current implementation is that all information is located on a relatively slow and inefficient database. Updates of information have to be hard-coded in the system. The system does not support access to other databases in or outside the current working environment. HyperCard is an event-driven, object-oriented programming environment based on the sending of messages from one object to the other. The top level is the HyperCard level which defines all the procedures and functions that are accessible by lower level objects. The next level of objects is the stack level. A stack can hold both processes and data. The data can be any combination of text, graphics and sound. A stack is a collection of cards which incorporate backgrounds, fields and buttons. Each of these are objects and can therefore send and receive messages. Objects can have a procedural execution sequence attached to them, and can have their visible representation set either on or off. Data associations can be obtained by using either static or dynamic linking. Links

may be established between cards in different stacks, regardless of the card's relative stack position. Bi-directional links can be programmed as well. These links are based on a unique ID, which is independent of data content and enables creating relations between data elements.

2. The SNAP II SFM subsystem environment

The Shipboard Nontactical Advanced Data Processing Program (SNAP II) facilitates development of computer support for the SNAP fleet. The existing logistic system, Supply and Financial Management (SFM) is an interactive SNAP II subsystem which supports requirements processing, inventory management, financial management, supply control, and Integrated Logistic Management (ILM) onboard SNAP II ships. It is a terminal-oriented system built on approximately 35 different permanent database files and about 120 different working areas (records). Each working area reads values from one or more files and updates one or more database files. The system is based on menus. Menus are hierarchical with tasks usually executed on the leaf level. Usually, during execution of a task, a corresponding working area is created and information is retrieved and displayed. When the task is finished, new or changed information in the working area is written back to the corresponding file(s). The SFM system provides the following basic functions (See figure 1):

- Supports requirements processing
- Inventory management
- Financial management
- Supply control
- Integrated logistics management

The SFM has two types of subsystem functions:

- User functions
- Service functions

User functions are directly generated by the user by selecting a menu on the screen or by pressing a key to perform an action. Service functions are those that are performed indirectly by other modules either inside or outside the SFM subsystem. The user can have access to either the Supply Menu or the Supply Customer Menu. The Supply Customer Menu provides access to a subset of the Supply Menu and is therefore not

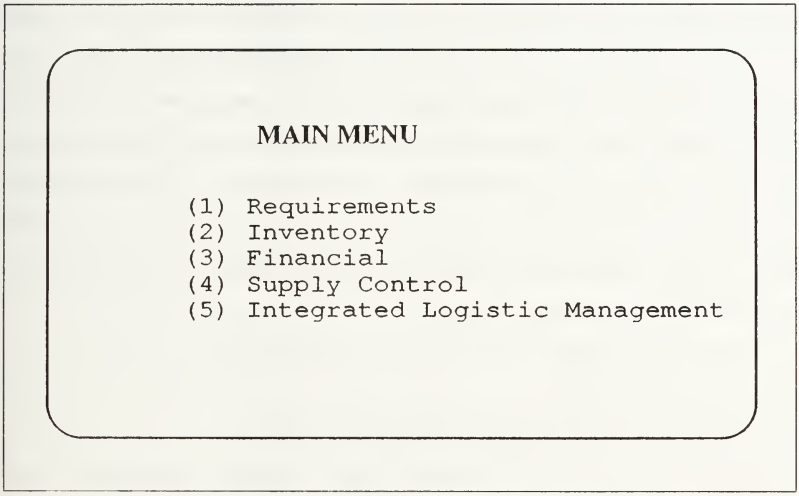


Figure 1: Building blocks of SNAP II SFM subsystem

discussed further.

SNAP II is written in COBOL and requires compilation for each change in the environment. SNAP II is running on 366 out of 540 required sites (as of July 1991.)

B. THESIS OBJECTIVES

A proposed system usually has a basic idea from which it originated. The general idea behind this proposed system is a mixture of the existing SNAP II SFM subsystem

requirements document and the general need for a more enhanced version making use of relational database management systems and easy to use features like a mouse, buttons and pull down menus. The original enhanced version (ARGOS) was developed on a Macintosh using Hypercard as a development tool [GIAN 89]. Hypercard provides an easy to use, object oriented, way of programming the Macintosh. Hypercard is an interpreted language with the abilities to store information as a combination of pictures, buttons, menus and table related plain ASCII text. A summary of the SNAP II SFM Subsystem specification is included in this thesis to provide general guidelines for the enhanced version. The SNAP II SFM system is written in a third generation language. Each change in a file structure, a record structure or the menu system requires a new compilation of all affected modules. The working area/file system model helps make the system modular and facilitates changes. Ad hoc[1] queries are only possible in predefined working areas. New views cannot be created without recompiling the system.

The enhancement of the system is based on two general ideas:

- Use of a relational model
- Separation of the user interface from the storage of data

This idea implies that all information including information about images, buttons, menus and functions are stored in the relational model. The proposed enhanced system uses a commercially available Relational Database Management System (RDBMS). All interfaces between the user interface module and the RDBMS use ANSI/SQL, when not explicitly expressed otherwise. This idea gives us the possibility to develop the user interface on a number of different platforms. Regardless of the user interface platform, the database information can reside on different machines in a distributed environment, on one machine accessed by all the user interface tools, or duplicated, on individual machines giving each user interface tool a separate database to work with.

---

1. Not predicted queries.

C. THESIS OUTLINE

This thesis starts with a discussion about different possible approaches to enhance the existing system. It gives the reasons for selecting a relational database management system approach. SNAP II SFM subsystem and the current ARGOS system guides as user requirements for the enhanced system. In addition to this CALS concept and MIL-STD-1388-2B's impact on development are discussed. An enhanced system based on relational database technics are proposed. Tables and procedures are developed for the enhanced system. Problems with a distributed implementation are discussed and a method for developing a distributed system is introduced. Finally a test system is developed and presented.

II. BASICS FOR BUILDING THE SYSTEM

This chapter describes the overall considerations taken for the enhancement of the existing system. It gives the reasons for selecting a relational database approach.

A. GENERAL

It is not possible to develop a system without looking into the tools available for implementation very early during the development process. Techniques for developing the system also have to be selected. Two possible basic approaches follow.

Use the state-of-the-art approach. While this is a hazardous approach, it has the benefit of giving you the latest available techniques. You can usually develop the system with less coding and with a shorter time to delivery. It will also give faster and more complex run time versions of the final product. However, there is a significant chance of ending up at a dead end. The developer has to be a visionary. If the selected approach turns out to be too sophisticated then it may never be possible to implement the system. Important decisions taken during the development could be based on the availability of products that never will reach the market. It can also turn out that you are the only buyer of the tools and supporting the system becomes unreasonably expensive.

The other approach is to use well-known techniques and products. The path to a working system will be well-known and very few surprises will slow development. The final system will not be state of the art but it will be reliable and working. Other users have tested the development tools before you and it is likely that these tools will stay on the market. The system may not give competitive advantage but it will be better than no system at all.

Small companies make small mistakes and large companies make giant mistakes. Bank of America tried to develop a state-of-the-art system and had to, after spending hundreds of millions of dollars, lower their expectations on the new banking system. The

U.S. Navy is an even larger organization. The possible customer to a new enhanced SNAP II system will be over 500 sites. By carefully selecting tools and techniques based on existing (de facto) standards and well tested technics, we can assure that our system will work, that it will be possible to enhance and improve, and that we will be able to interact with other systems. The Navy and other government agencies are filled with non-intractable systems.

B. INTERFACE

In most existing systems the user interface is built as an integrated part of the system. Changing the user interface usually involves so much effort that it is not worth changing without also taking the opportunity to change the rest of the system as well. Different people accessing the same information require different user interfaces. A low frequency user will need more 'user friendly' features like menu choices, point and click options and intuitive access to information. A frequent user is usually more concerned with speed and direct access to information. A person working with the system on a daily basis does not want to have to go through a hierarchy of menu choices each time he accesses information. He would prefer to use a few keystrokes to take a lot of action on the stored information. By completely separating the user interface from the stored information it is possible to develop a system that:

- Can have different user interfaces accessing the same information.
- Can enhance and change the user interface separate from the stored information.
- Is platform independent both with respect to the stored information and the user interface.

Separating the user interface from the stored information requires a well defined set of tools available to interact between the two. The only widely used language to access information on database management systems today is SQL[1]. By selecting SQL as the

1. Structured Query Language, language developed by IBM for accessing database information, pronounced 'sequel.'

interfacing language the system will, at least in the near future, have to be based on a relational database management technique. There will probably be products available in the future that give access to other types of information storing systems using SQL.
Every transaction between the user interface and the stored information must be a SQL query in one direction and the result of the query in the other direction and nothing else.

C. RELATIONAL DATABASE MANAGEMENT SYSTEM

Relational database technique is today a well defined science with a number of available tools for managing the information. It has a defined algebra for calculating the result of accessing information and a query language to communicate with the database management system. Available tools have been on the market for a number of years and are therefore well tested and debugged.

The relational database should be perceived [ORCE 88] by its users as a collection of tables (see figure 2). Tables consist of rows and columns. The intersection between a row and a column is called a field. A field contains a value. Other requirements for a Relational Database Managements System are:

- Existence of a high-level language.
- Full relational processing capability.
- Maintainable as a system.
- Integrated data dictionary.

The relational database technique is widely used today and some of its most well-known developers include ORACLE, INGRES and INFORMIX.

D. DISTRIBUTED SYSTEMS

There are both advantages and disadvantages to using a Distributed Database Management System (DDBMS) compared with a Centralized DBMS(CDBMS). A typical distributed system is shown in figure 3. Some of the most important advantages are:

- Improved reliability. If the information is stored in different locations it is more likely

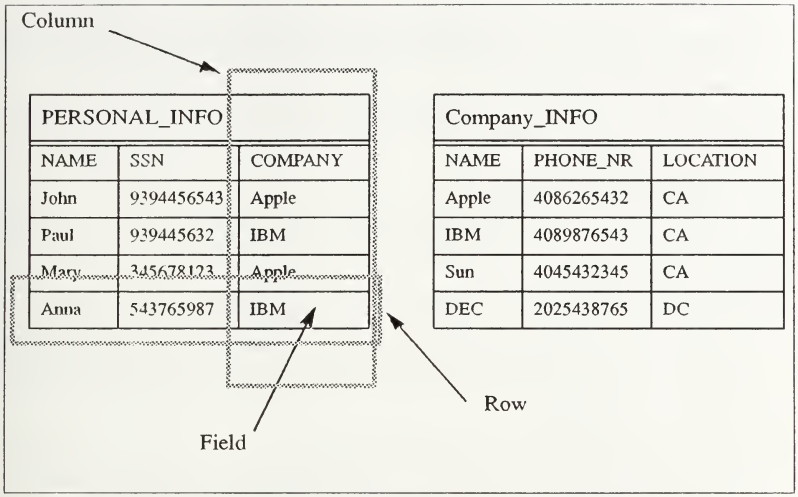


Figure 2: Tables, the building block of a relational database

that the information is available when it is needed. If one site goes down, most of the sites will still be able to run the system.

- Expandability. It is easier to expand a database locally without changing the other sites.
- Improved performance. With information stored on different sites, there will be occasions when processing can be performed in parallel. I/O processes will be executed in different locations. This will be a trade off with the cost of communication between the sites.
- Local autonomy. Locally stored and shared information could be handled locally without affecting the rest of the database.

Some of the disadvantages are:

- Lack of experience. There are very few running examples of truely distributed databases.
- Complexity. All complexity in a distributed system is inherited from the centralized system. In addition to that we have to add the complexity of distributing the

information.

- Security. By having a centralized database we can much more easily control access to the data stored in the database. In a distributed system we have a more complex network and storage schema.
- Cost. Having databases stored on different sites multiplies the cost of managing the database on each site.

1. Special problem areas

There are some special problems with the DDBMS that are not yet solved, but where research is ongoing:

- Distributed database design. Partitioned data, replicated data or partially replicated data are some of the design issues that have to be considered for the DDBMS. There are a lot of mathematical problems that have to be solved in order to minimize the cost of storage and communication. There are not yet any design methods that help a developer in doing this.
- Distributed query processing. Algorithms that execute a query in the most efficient way must be written. The objective is to distribute the query to achieve as much parallelism as possible and at the same time limit the network use. Usually those kinds of problems are NP-complete problems.
- Distributed concurrence control. This is the most studied part of DDBMS problems. It involves questions like: How to maintain consistency in the database?

The highest level of interaction could be described as the Distributed DBMS. This level of information interaction is not even close to standardization. So many unsolved questions still remain that it is unlikely that a standard for a fully distributed system will be developed during the next 10 years.

Some examples: SQLnet, for the Macintosh, from ORACLE supports access to only one server at the time. Access to different servers is not transparent to the user. There are server-server connections that exists for both INGRESS and ORACLE but those can only be accessed from a client, with the same restrictions as mentioned above. So why can DBMS vendors claim that their product is a DDBMS? And why can they claim that their databases

in a distributed system are transparent[1] to the user? Well, they are assuming that no fragmentation and no duplication of information is made without the user writing all of the code necessary to do this. If the user wants to have the information duplicated and/or fragmented, all of the necessary procedures to do this, like the two phase commit protocol[2], have to be written by the customer. There are vendors(SYBASE) that claim that they have implemented two phase commit (2FC). But there is no standardization of the protocol for 2FC. So even if two DDBMS's have 2FC implemented that does not necessarily imply that the implementation is compatible with other vendors' 2FC protocol. ANSI SQL has not yet

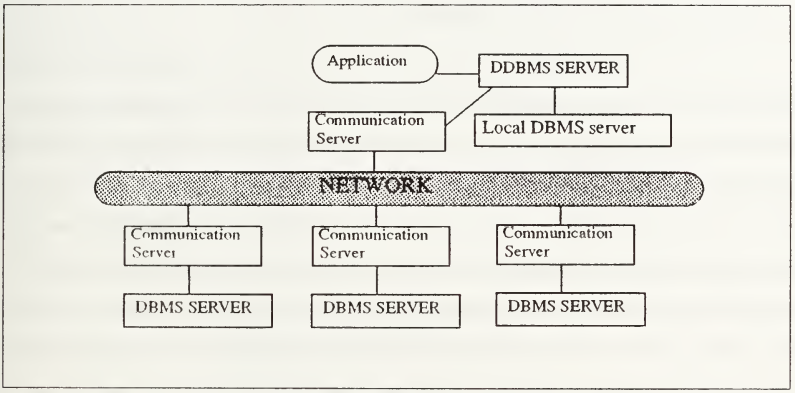


Figure 3: The distributed system

defined any syntax for manipulating information among different sites.

SQL Access Group SAG [SAG 91] was founded 1989. SAG is a non profit group of vendors that are working together to create an SQL standard beyond what ANSI/SQL provides. A group of companies including Ask/Ingres, Tandem Computer, Sun

1. The user perceive the distributed database as one database residing on his own machine.
2. Protocol that guarantees consistency in the database among sites.

Microsystems, Digital Equipment Corp and Hewlett-Packard started meeting informally to discuss how to make it possible to interact between different DBMSs.

The SQL Access Specification is based on existing standards like ISO/RDA and ISO/SQL and from guidelines like X/Open Company Ltd. The SQL Access specification is included in the 1991 edition of the X/Open Portability guide. SAG has concentrated on:

- Association Management Statements.
- Standardized System Catalog Names.
- New Catalog Tables.
- Diagnostic Messages.
- Conformance Levels for SQL Implementations.

SAG had an interoperability demonstration on July 16 1991. A number of Client and Server applications where working together over a network. All running applications were built on the technical specifications in order to test and verify them. If SAG succeeds with it's mission, user applications can be developed independently from the DBMS server. A client can access any DBMS or gateway that follows the SQL Access Specifications. This is a very promising future, creating a lot of new aspects of accessing information on DBMS. In the ARGOS project this will imply that the user interface, written according to the SAG specifications, will run together with all DBMS systems using the SAG specifications. The above problems restrict the use of a distributed system. This enhanced system uses an expanded SQL. This expanded SQL does comply with the naming conventions used by SAG.

E. OBJECT ORIENTED APPROACH

A fairly new idea for implementing the data base system is using an object oriented approach. The following description compares differences, advantages and disadvantages between using an object oriented approach for implementation and a traditional relational

database approach [TARI 92]. It also describes some of the most important concepts behind object oriented implementation. This includes:

- Encapsulation
- Classification
- Inheritance
- Aggregation
- Polymorphism

As implied by the name, the foundation for an object oriented view is the object. Objects includes both information (data) and behavior (code). This is in contrast to a relational database where the corresponding foundation is the field in the table which can only hold information (value).

Encapsulation. The view of the object always includes both information and behavior. An object can change state without changing the information in the object. A message can be passed to an object causing a change of state. A message can be passed back as a result of the change of state.

Classification. By grouping objects which share some attributes a class is created. This implies that an object is an instance of an class. Classes are organized hierarchically. In a relational database system the table can be viewed as a class.

Inheritance. Classes inherit all attributes and behaviors from their ancestors. Class inheritance comes from both direct and indirect ancestors. If a class inherits attributes from a single direct parent it is called single inheritance. Multiple inheritance implies that the class can have multiple direct ancestors. Inheritance does not exist in relational database systems.

Aggregation. Attributes of an object can form aggregation of other objects. These can be simple or complex objects. Simple objects are of type integer, string etc. Complex objects can include attributes that form aggregation as well. Aggregation is a way to describe the sharing of data and behaviors. In some application it is possible to change

aggregation dynamically.

Relational database systems can not handle aggregation. Changing attributes is not normally considered possible. Though, creating a view could be in some sense like aggregating data.

Polymorphism. This term is used to describe the concept that objects from different classes can invoke the same method. This implies that when an object receives a message it can take many different forms. This is important in the context of reusing software. Polymorhism does not exist in a relational database system.

An object oriented approach introduces many nice features that do not exist in a relational database management system. A state of the art system, created with object oriented methods would probably be more efficient than a traditional relational system. A major difficulty with introducing object oriented systems is that no defined standard yet exists. It is possible to end up at a dead-end.

F. OTHER APPROACHES

A federated approach. Postgres, the successor of Ingres, is an example of a relational system with object oriented components. Information in the database can have attributes forming behaviors. The concept of class and inheritance is also under development. This is an even more state of the art approach than the true object oriented approach.

Efficiency, or rather lack of efficiency, could be a reason for leaving the relational database management system approach. If distributed technics combined with a compiling relational database management system is not fast enough, then a move to a traditional compiling language maybe the only solution to the problem. This could be an initial development on a relational system. It would take advantage of early prototyping capabilities and, when the system is almost developed move over to a third generation language.

III. USER REQUIREMENT

This chapter describes the user requirements that are used as a guideline for the enhanced system.

A. GENERAL

User requirements are based on existing systems (SNAP II) and proposed standards (MIL-STD-1388-2B). When this is not applicable, requirements are based on general reasoning.

B. SNAP II

The user requirements description is based on the existing SNAP II SFM subsystem [SNAP 86]. It serves as a guideline for the enhanced system. The following description is based on the first menu in the SFM subsystem.

1. Requirement processing

By selecting the Requirements Menu from the Supply Menu the user gains access to a number of functions for managing single line items. This includes input, issue, ordering, status tracking, material receipt and requisitioning. All actions center around a single line item. An item can be identified by its stock number, part number/FSCM or by Money Value Only data. Basically, all actions from the discovery of a need for a new item, throughout the whole life time of that item could be monitored. These include:

- Material request.
- Approve/reject request.
- Forward request.
- Material transfers (turn-in and issue).
- Issuing of reversal.

All transactions that are necessary for the item's history are recorded. By doing this it is possible in the SFM system to monitor the status of a requisition. The request can

be partially changed or rejected before it gets approved. Once the request is approved it stays in the system. Approved requests that are not performed cannot be deleted. If an item is changed between the time it is requested and it is received, necessary changes can be made. A wide variety of reports can be accessed through the requirements menu:

- Print all 1250 request documents.
- Print all requests.
- Print all not approved requests.
- Print all completed requisition.
- Print all outstanding requisitions.

It is usually possible to restrict the listings by certain keys including workcenter, dates and status.

2. **Inventory**

The Inventory Menu, selectable from the Supply Menu, enables the user to monitor the number of items in stock. The system provides the user automatic reorder processing. Reorder suggestions are based on high and low allowances in stock and can be changed by the user. Each reorder has to be approved before it goes into action. Reorders can be made on items individually or by groups. The inventory system handles both Coordinated Shipboard Allowance List (COSAL) items and demand based items.

3. **Financial management**

The Financial Management Menu provides the user with functions for Operational Training and Readiness (OPTAR) funds management and issuance of financial reports. The user can make obligations and grants adjustments and produce various reports. The system also handles reversing obligations. If an obligation is reversed the appropriate values in the financial files are changed. Obligations are identified by their request or requisition number. Obligation adjustments can be made by changing the fund code and/or amount. If the request does not exists an AUOL may be processed. The AUOL screen gives

the user access to enter information such as Requisition Number, Cosal Type, Fund Code, Document Identifier, Priority Code, Unit of Issue, Quantity, Extended Money Value and Requisition Completion Date. Producing a new budget and periodic updates are not provided by this menu. Reports include:

- Summary Fund Code Difference.
- Budget OPTAR and Summary.
- Financial Transmittal.

4. Supply control

This menu is the Supply Officer's tool. By accessing this menu the Supply Officer can manage new users. He can change the access level on users and create history files, update COSAL types and handle repair parts management. When adding a new user the Supply Officer has to provide the access level for the user including:

- Authority level.
- Deleting capability.
- Menu depth.
- Work center.
- Division.
- Department.

The access level for each user is stored and menus are displayed in accordance with the specific level of access. The supply officer can also perform limited actions on the Budget Record.

5. Integrated Logistic Management(ILM)

This is the highest level of access. It is given to the Subsystem Manager. Some of the menus under this menu can be assigned to different subordinates. New configurations are created through this menu. Initiation of the new budget year, adding new accounts, changing General Access Levels, and Technical Documentation Management are all performed in this menu. Defining a new Inventory Section includes both defining the

section and defining the items belonging to that section. This action implies that the user must be able to change the existing sections as well. When inventories are relocated appropriate reports are printed. No records for previous locations are maintained. A very wide variety of reports including calculations are accessible from the Integrated Logistic Management(ILM) level:

- Inventory Reports.
- Inventory Work sheets.
- Shortage Calculation Reports.
- Single Item Release Document.
- Material Receipt List.

Other important functions are:

- Add/delete Allowance Parts Lists(APL) to/from critical Equipment list.
- Security management of subordinates for accessing different ILM menus.
- Double Clerk inventory management.
- Establish ship in Integrated Logistic Operational(ILO) report.

C. REQUIREMENTS FOR A LOGISTIC SUPPORT ANALYSIS RECORD

MIL-STD-1388-2B [MILS 91] describes Data Element Definitions (DED), data field lengths, and formats for Logistic Support Analysis (LSA) record (LSAR) data. DOD contractors are required to use this standard to provide physical and logical breakdowns of delivered system and subsystems. The standard is described and discussed further in 1.Existing relations in MIL-STD-1388-2B on page 26.

D. CALS CONCEPT

CALS stands for Computer-aided Acquisition & Logistic Support. CALS is a Department of Defense initiative that addresses the need to transition from the paper intensive handling of information to handling based on digital data [MOLI 91]. The idea covers the entire life cycle of a weapon system from design, through use and disposal.

CALS goal is to lower the overall weapon systems cost. Some of the most important issues are:

- Increasing productivity in design, thereby decreasing lead time.
- Lowering costs of program management through streamlined processes.
- Improve quality of weapon systems, including increased reliability.

What CALS is trying to achieve is the same as what corporate America is trying to achieve, exchanging information in digital form to increase efficiency and lower cost. The CALS initiative involves not only the DOD but also a number of large and small companies involved in military development and production. CALS does not invent new standards but rather uses well-known and well tested existing standards. The most important CALS accepted standards today involve definitions for storing and interchanging images and standards for writing technical manuals including a mark-up language. By complying with a parent standard all child standards must be used as well (see figure 4).

Since the CALS concept involves all logistic information related to the lifetime of a weapon system it also addresses storing information in relational form in databases. No standard for storing information in databases is agreed upon yet but it is most likely that the MIL-STD-1388-2B will be accepted by CALS.

The CALS standards that affect the proposed system in this thesis are:

- MIL-D-28000 - IGES vector data.
- MIL-R-28002A - Group 4 raster data.
- MIL-D-28003 - CGM vector data.

These standards define the format for storing images. All of them are graphics standards and provide definitions for storing images in raster format or as vector objects. Vector objects are not useful for identifying real objects like components, implying that we can only use these standards as image storing formats and not for storing relations between components and subcomponents.

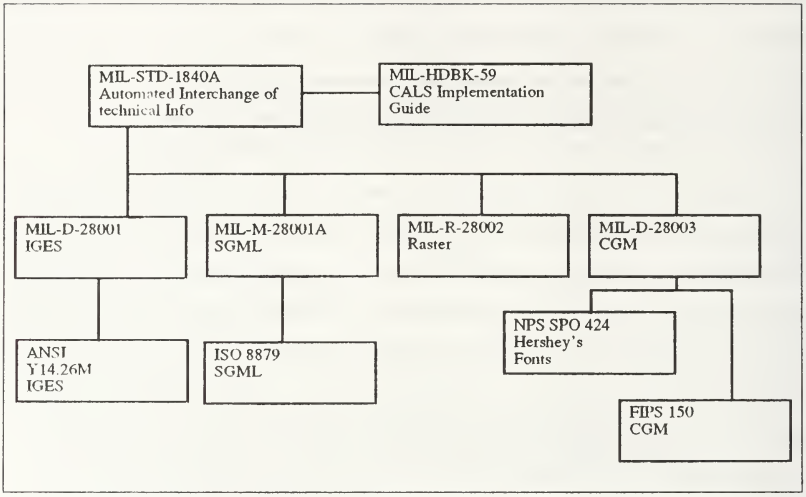


Figure 4: Some of CALS standards

E. ARGOS CONCEPT, THE PAPERLESS SHIP

1. User interface

The user interface should work in the same manner as the existing system. The top level screen displays silhouettes of the ships that form the battle group (fig 5). At this stage the user selects a ship by pointing to it with the mouse and clicking.

To retrieve information concerning an item related to the selected ship, the user has to navigate through the ship's structure until he finds the item he is searching for. For example, if he is interested in the ship's engine he would navigate through the ship's structure (ship -> engine room -> engine (figure 5) until the engine is displayed. If he desires to go deeper in the search he can select the engine and an exploded view of the engine will be displayed (figure 6).

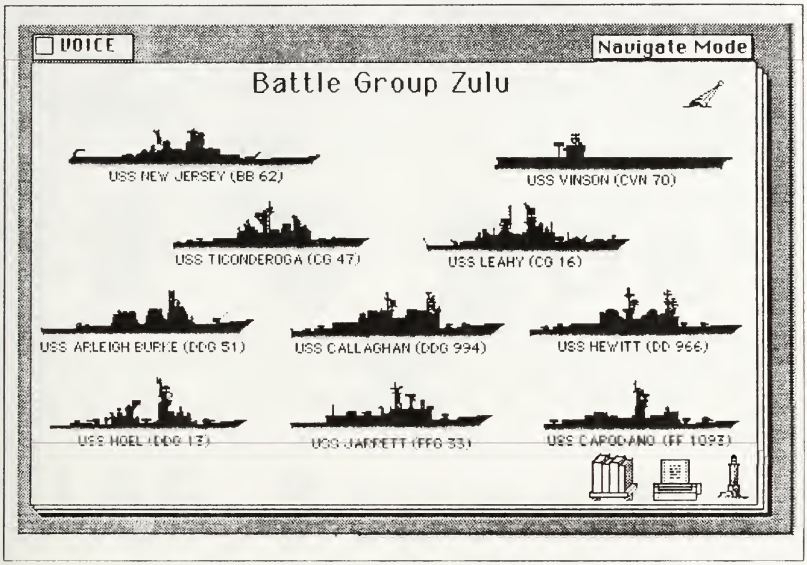


Figure 5: Initial screen

At this point the user has the option to retrieve Allowance Parts Lists (APL, COSAL and/or Equipment Identification Code(EIC) information about that item (the Gas Turbine Engine). If he wants to order this item he clicks on the order button and an order form will be displayed. If he wants to order a subpart (High Pressure Turbine Nozzle) he has to select the item, resulting in a detailed image of the item being displayed. Thereafter he can order that specific subpart.

When the order form is displayed the system will fill in all the fields that have corresponding values in the database (figure 7).

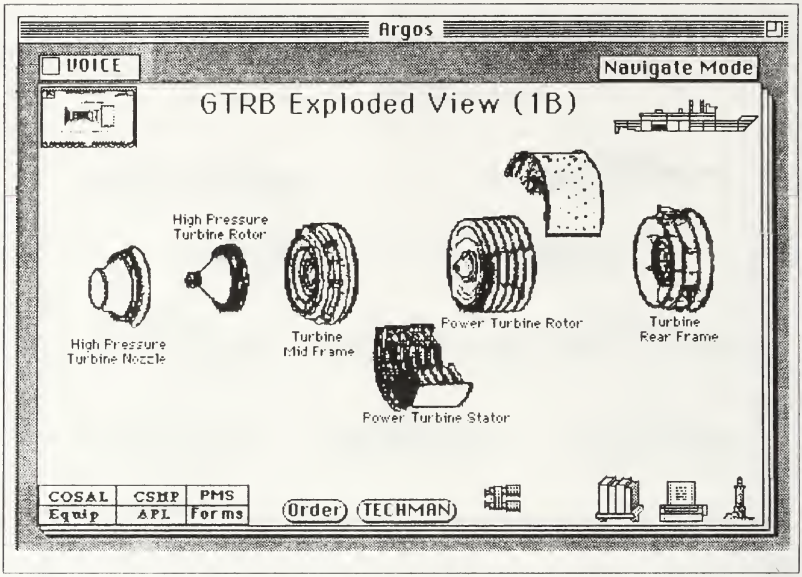


Figure 6: The user has found the item he is searching for

The implementation of other modes of operation of the system is handled in a similar manner.

2. Functional specification

The following specifications shall apply:

- The user shall be able to insert a new item in the model and link that item to the ship. If there exists corresponding APL,COSAL or EIC information the system shall be able to create, given the link data from the user (i.e. the corresponding APL Key, etc.), a link to that information.
- The user shall be able to delete an item from the model. When deleting an item the system shall delete all links that are associated with that item. Only the specified item is deleted, leaving all sub items intact. The user shall also be able to delete a link to an item, leaving the item in the model untouched.

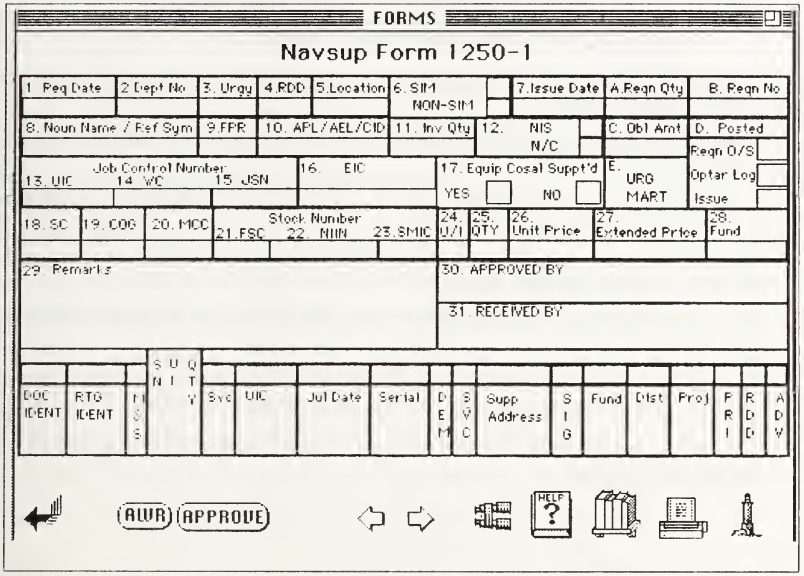


Figure 7: Form for ordering an item

- The user shall be able to change an item. This could be performed by changing links, changing images, changing buttons or changing fields.
- The user shall be able to navigate through the ship's structure.
- Buttons will be displayed only if the execution implied by selecting them is meaningful/possible at that level.
- From each display the user shall be able to retrieve the corresponding APL, COSAL and EIC information, if such information exists.
- From each display the user shall be able to change the corresponding APL, COSAL and EIC information, if such information exists.
- From each display the user shall be able to delete the corresponding APL, COSAL and EIC information, if such information exists.
- The user shall be able to insert new APL, COSAL and EIC information.

3. System requirements/constraints

The following requirements and constraints apply:

- The system shall operate in the current user environment (i.e. no organizational changes will be needed).
- The system shall not propose any changes to the current structure of the APL, COSAL or EIC databases.
- From the point of view of the user, the APL, COSAL and EIC databases will continue to exist as separate databases.
- Buttons, fields and image pointers should be stored as a part of the local changes to the APL, COSAL and EIC databases.
- The system shall support storing APL,COSAL, EIC and images on a permanent media (CD-ROM).
- The system shall support storing local changes without changing the original data.
- The system shall support accessing SQL databases outside the working environment.
- The system shall be defined so that it is possible to develop the user interface on different computers.

IV. RELATIONS, TABLES, AND PROCEDURES IN THE ENHANCED SYSTEM

This chapter describes the relationships between the stored images, buttons and procedures and the existing stored information. It also describes procedures (pseudo code) necessary to manipulate the information.

A. GENERAL

It is important to keep the logical relation in the existing systems, regardless of whether that system is the existing SNAP II SFM subsystem or a future system complying with CALS specifications. DOD MIL-STD-1388-2B is assumed to be the relational structure for database information. SNAP II is not in compliance with SLAR standards so relationships, in the proposed system, are described both with respect to SLAR and to SNAP II. Relations to this information are described in the relational tables below. For consistency reasons the SLAR definitions for items are used.

Assembly: A number of parts or subassemblies, or any combination thereof, joined together to perform a specific function.

Attaching part: An item used to attach assemblies or parts to each other or to the equipment.

Component: An assembly or any combination of parts, subassemblies capable of independent operation in a variety of situations.

End article/Product: A component, assembly or subassembly being procured as the top item of the contract.

End item: A final combination of end products which is ready for it's intended use.

Some extended definitions are introduced.

Individual is defined as something that for one or more of the following properties is true.

- Its parent is an item (works recursively).
- Its child is an item (works recursively).
- It can be displayed.

- It can be ordered.
- It has a price.
- It can be counted.
- It is an inventory.

This implies that the set of objects in the MIL-STD-1388-2B are a subset of individuals. All objects in the MIL-STD-1388-2B are individuals but not vice versa. An individuals's only reason for existence can be to relate objects in the standard to each other.

B. RELATIONAL SCHEMES

1. Existing relations in MIL-STD-1388-2B

The existing standard is based on the idea of storing one structure of a system. It is created for the purpose of storing the structure in a digitized media. The standard has the flexibility to add information about new and alternative versions of components. It provides relations to attach additional general information to components. This information includes stock number, prices and similar information. There are also tables to store statistic information about the system and its components The relation between a parent component and a child component is hidden in the Logistic Support Analysis Control Number(LCN) (figure 8). The upper limit of numbers of alphanumeric characters in the LCN field is 18.The numbers define the relations. In figure 9 the A in the LCN code stands for the overall system, the TRUCK. As the system is broken down further the number of used alphanumerics in the LCN field increases. By looking at the value in the LCN field it is possible to see the parents and grandparents all the way up to the root of the system. The Alternate LCN Code (ALC) gives the ability to store alternative designs. For example two different engine alternatives. Usable on code (UOC) are used if specific items are used in more than one place. This does not apply if the items are the same but require different maintenance or differ in other important aspects. There is a possibility to create both physical and logical system break-downs in the MIL-STD-1388-2B, see figure 10. An

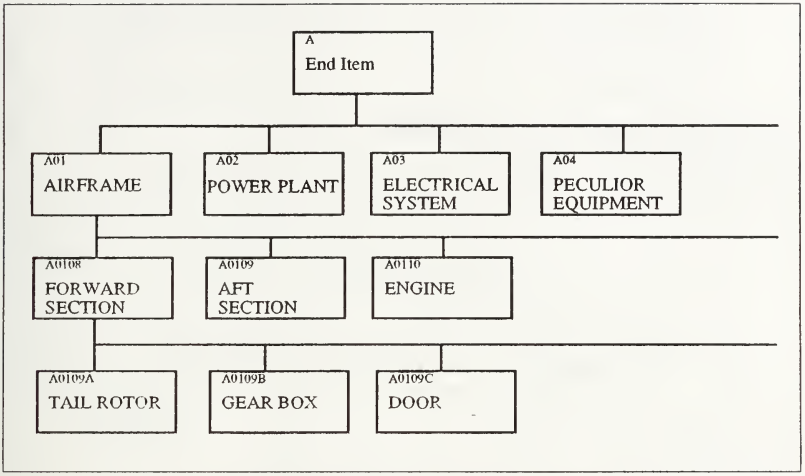


Figure 8: Classic LCN assignment method

antenna has the physical location on the turret, but logically it belongs to the communication system. In that case the item related information is stored only once.

The standard also provides a guideline for developers, who have the responsibility to create and provide the relational data together with the product.

One drawback of the MIL-STD-1388-2B is that it does not provide relations for identifying unique objects. Assume, for example, that we want to identify a system and attach a history to a component. Every time something happens to the system or a component in the system, we want to log that event and attach it to the correct component. Nothing in this standard supports this concept. Another questionable construction of the standard is the way relations are described. All relations are based on the numbers in the LCN and ALC fields. This restricts the number of levels that could be used to describe the system. It also makes it difficult to change relations by inserting, or deleting levels. Two parent relations are

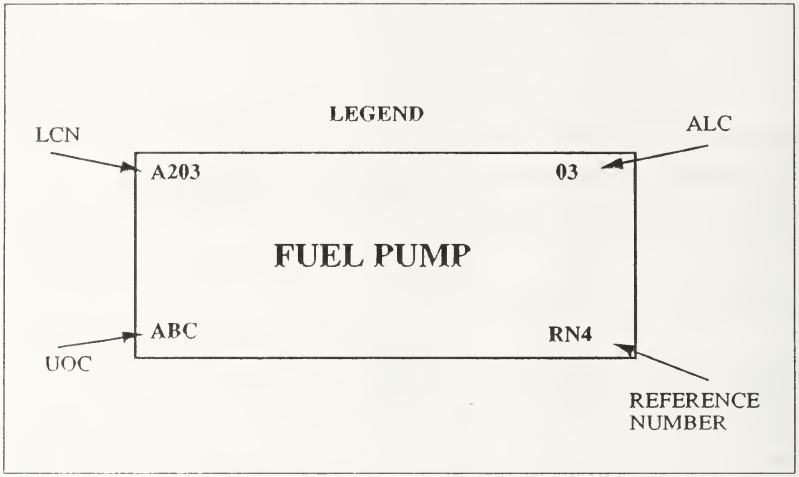


Figure 9: Item key fields in the MIL-STD-1388-2B

possible only when creating two different breakdowns. The standard does not directly support storing images together with the text information in the fields.

At this point it is clear that some improvement have to be made in order to support the requirements. Possible enhancements are discussed in the next paragraph.

2. Basic building blocks for the enhanced standard

The small system in figure 11 is used as an example for describing the enhanced system. This example is a simplified entertainment system onboard a Navy ship. It consists of a broadcast unit, two small room units, one large screen room unit and attaching wires. The attaching wires, both in the room units and between the room units, together with the TV's (but not the VCR's) and a Commanding Officer (CO) broadcast unit are a part of the entertainment system as well as part of the CO broadcast system. All the wires are also a

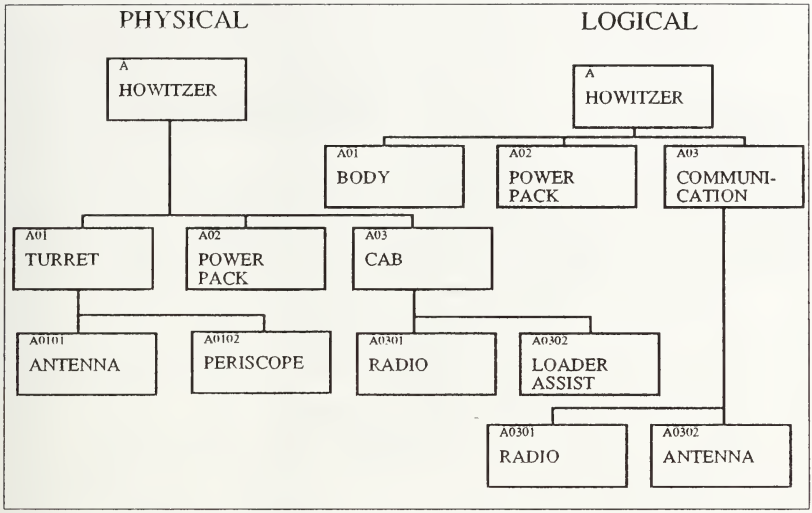


Figure 10: Physical and logical breakdown

part of the wiring system, so there is actually an almost unlimited number of systems that can use the components in our system.

The most important part for the functionality of the system is how the relation between a parent and a child is described. Since this thesis is using the relational database technique as a preselected technique for storing information, other techniques like hierarchic, network and object oriented are not discussed. Relations between a parent item and a child item can be described only in one way if the system is to take advantage of the flexibility of the relational database management system. There are two different columns (figure 12), one called parent and one called child. New relations can be added by adding a new line in the table. Relations can be deleted by deleting a line in the relational table. Only the available storage on the system limits the number of relations that can be

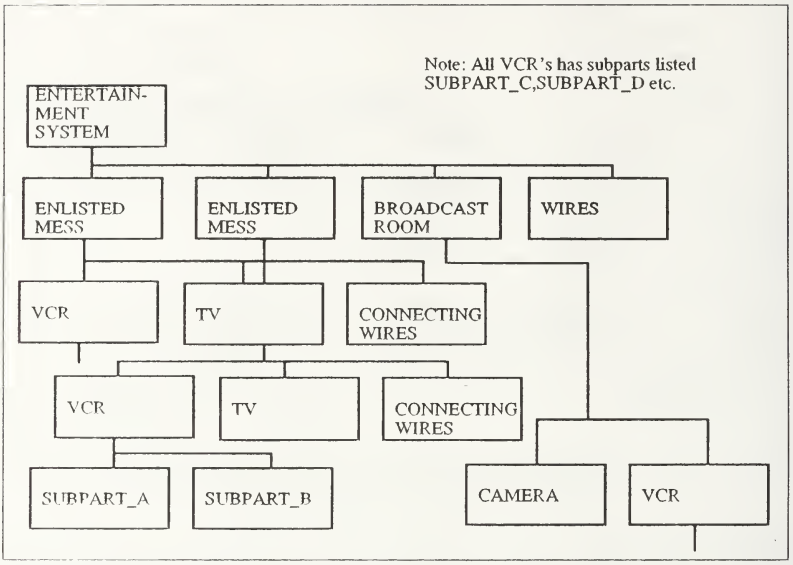


Figure 11: Structure used to describe the enhanced system

described. One table could be used for all systems (this could be restricted for other reasons). The techniques used in MIL-STD-1388-2B, using the LCN code, are lacking the flexibility that the parent-child relation provides.

Some of the most important requirements affecting the building of the system are:

- Flexible relations.
- Multiple parents and cross relations.
- Possibility to identify individuals or default to basic type information.
- Possibility to relate images.

The first two requirements in the bullet list are solved by using the child-parent relation. The third one is more difficult.

PARENT-CHILD	
PARENT	CHILD
Entsyst	Enl_mess_1
Entsyst	Enl_mess_2
Entsyst	Broad_cast
Entsyst	Wireset_1
Enl_mess_1	VCR_1
Enl_mess_1	TV_1
...	...
VCR_2	Subpart_F

Figure 12: Parent-child relation in a relation table

Assume the initial system is described in our tables as a basic system without individuals. This implies that our VCRs are only stored once. The parent-child relation from the two rooms containing the VCRs relates to the same object (see figure 13). The task now is to create additional relations so that the VCR's become individuals that can have attached information. Since the rooms where the items are stored are not individuals they must become individuals as well. This principle applies all the way up to the root of the system. Creating an individual from a VCR does not make sense without having defined the ship, the system and the rooms the VCR's are in. If the VCR becomes an individual it does not imply that it's sub components have to be individuals as well. Every individual in the database has to be defined as a relation to its parents. This is also necessary for other reasons like counting the number of items of a particular identity. What about different

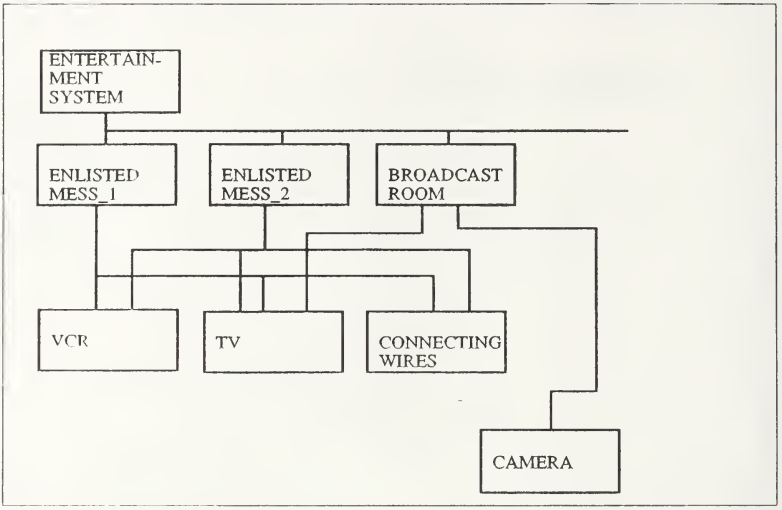


Figure 13: Different parents pointing to the same child,
thus beeing different individuals.

parent systems? Do they have to be individualized as well? Let's look at an example. As shown in figure 14, a TV stored in the Entertainment System is stored as two individuals in the CO Broadcast system. This indicates an inconsistency since we are referring to the same object even if it is stored as different objects. If we would like to change TV1 to an individual it has to be done on both the CO Broadcast system and the Entertainment system. This is not satisfactory.

A better approach is to reflect reality in the initial relational table, letting each object be an individual. The overhead for doing this is minimal compared with the complexity of changing the objects each time an individual is created. A conceptual view of this example system is in figure 13. The biggest advantages of this system are:

- Reflects reality.

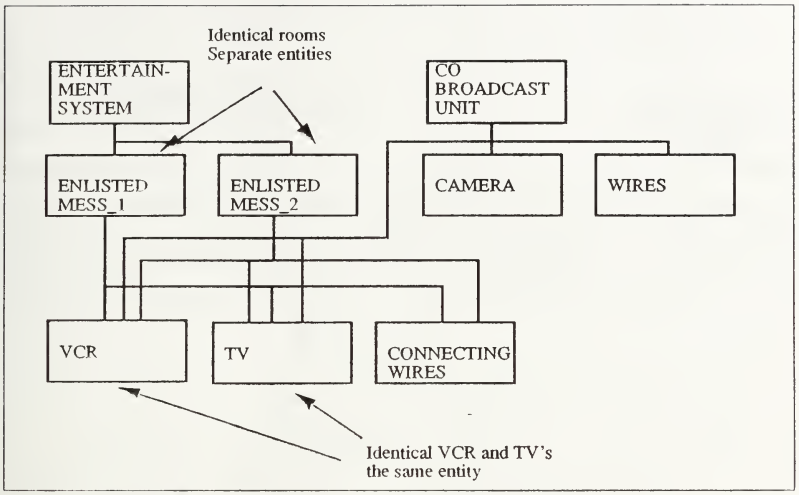


Figure 14: Inconsistencies in multiple parent systems.

- Intuitive.
- Less run time changes of relations when creating individuals.

Can this idea handle changes and describe alternative designs? Since the basic idea of the system is to reflect the real existing system a change of a subcomponent will be reflected by changing the relation. This thesis is not explicitly addressing alternative designs. One direct method of doing this is to create a different relational table for an alternative design. The problem with this method is that with many alternative designs there will be a "lot of" tables. A better method is to store alternative designs in the original relation. This requires an additional column in the parent-child relation showing which alternative design is being used. This must also be reflected in the display so that the user can select the desired design. Relational diagrams and Tables for this system are shown later in this chapter.

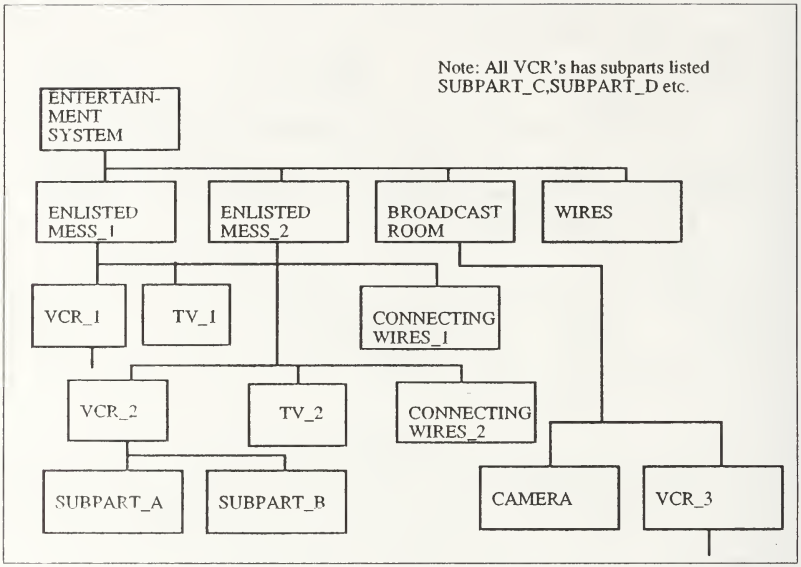


Figure 15: Conceptual view of the final system

The forth and last requirement on the list is to include images in the system. The system shall be able to handle images showing the different components in the system. The system shall also be able to handle a point and click method to traverse the information. Assume that we have an image showing the Entertainment System. Each of the objects on this image must have a button on the image for each subcomponent (child). A list of subcomponents could be substituted for the button. The ideal is to have an image where the buttons are located above the displayed subcomponents (invisible). This requires that someone (developer?) relate the subcomponent buttons to the correct place on the image. Since a subcomponent can be displayed on more than one parent image, the same

subcomponent must be able to relate to more than one button and more than one image. The same component image shown on the display, can be related to different real objects. Compare the TV's in the different rooms (figure 14). This points to a solution where the images do not show the relations as in the existing ARGOS system, but only reflect what are in the tables describing the parent-child relations. The unique information of an object is reflected in the parent-child relation so the button location will naturally go in as an extra column in that table. This creates overhead since some of the images are the same. This overhead is acceptable since the alternative, creating an image-button table, wastes storage.

3. E/R DIAGRAM

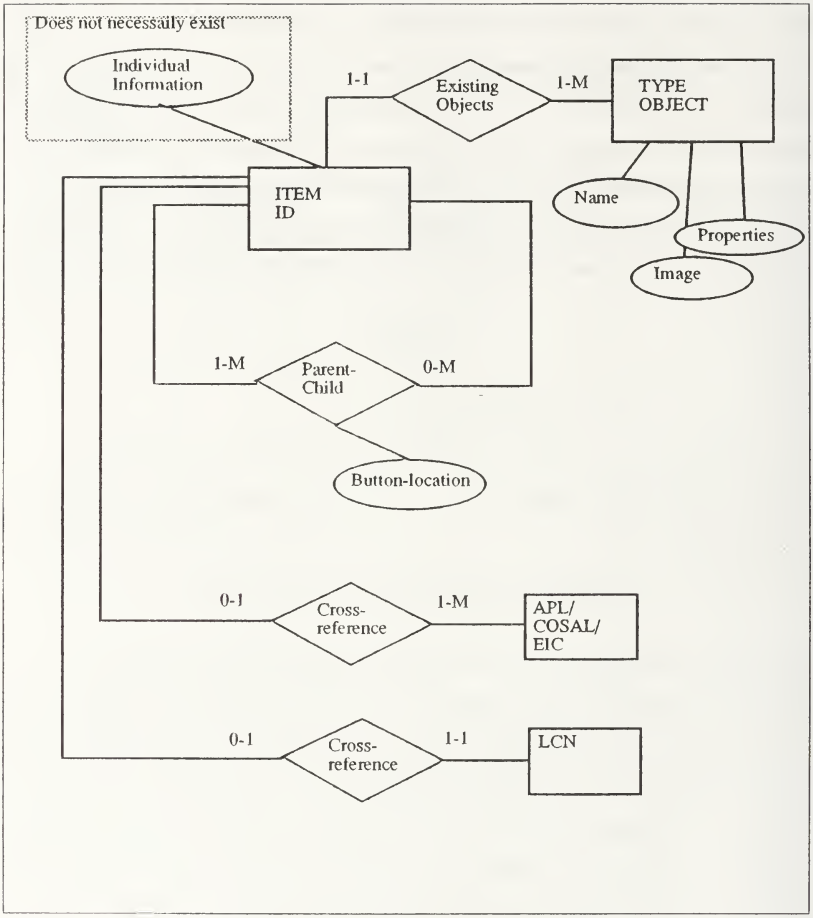


Figure 16: E/R diagram for the basic system

C. TABLES

Every time the user is traversing the system the PARENT-CHILD table (table 1) is used to move from a Parent ID to a Child ID. The Child ID then becomes the new Parent ID. The Parent ID is used in the ID-REAL table (table 2) to relate to a specific object (Type_ID). Neither the ID nor the Type_ID numbers have to be meaningful to the user. They are only used for relating objects. The Type_ID number is used in the SPECIFICATION table (table 3)to retrieve the actual image and to store general information about that component. It is not necessary to use the INDIVID table for storing the existence of an individual if no individual information is attach to the individual. This is under the assumption that an object without a parent-child relation does not exist. In the proposed system all objects are defined as a tuple in the INDIVID table (table 4)as well as in the ID-REAL table even if no individual information is attached. When an image is displayed the ButtonLoc for all Parent ID's is retrieved and stored on the image together with Child ID for that button.

1. Parent-Child

Field description:

Parent ID: Used to identify an individual. Unique for the individual and generated by the computer.

Child ID: The same as Parent ID

ButtonLoc: Describes the location of the button on an image. Described as upper right and lower left corner as a percentage of the overall length and width of the image.

PARENT-CHILD		
Parent ID	Child ID	ButtonLoc

Table 1: The parent child relation table

2. **Type - individual relation**

Field description:

ID: Used to identify an individual. Unique for the individual and generated by the computer.

Type_ID: Many individuals can relate to the same type of object. This is an one-many relation implying that the table could be replaced by an extra column in table 4.

ID-REAL	
ID	Type_ID

Table 2: Relation between the individual objects and Type_ID

3. **Type related information**

Field description:

Spec: Many Id's can relate to the same type of object. The Spec is the code for an object type (like a specific TV type).

Name: The specific type object has properties. Name is one of them.

Image: Stores the information about the location of the image (could be the path to the image).

Other spec: Columns can be added for each of the additional information items that need to be stored about the objectField description:

SPECIFICATIONS			
Spec	Name	Image	Other spec

Table 3: Relations between real objects, images and descriptions

4. Individual related information

ID: Used to identify an individual. Unique for the individual and generated by the computer.

Status: One example of individual information.

Textpointer: Can be the path to a block of text describing the history of the object.

INDIVID			
ID	Status	Textpointer	Other info

Table 4: Stores individual information about objects

D. PROCEDURES

This section describes a selection of procedures in the enhanced system which perform different actions on the database. All procedures are described with pseudo code. Access to information in the database is always executed using ANSI SQL [ANSI 89]. Procedures are also described based on a user interface that allows users to point and click on images. When an action (order, change status, delete etc.) is performed on an item, it is performed only on the item displayed. This gives two different approaches for the user interface. First, the exploded view of an item serves as the substitute for the large image on the selected item. This approach will speed up the traversing of the system. Second, when clicking on an object, an enlarged view of the object can be displayed. This enlarged view gives access to perform actions on the actual object. One action is to explode the object. When the object is exploded the user can continue to traverse the system. By selecting the second approach some additional problems are introduced into the system.

- All objects are not individuals or one individual has more than one image attached.
- If the system allows both methods inconsistencies occurs in the system.
- If the system does not allow both methods many extra images must be created.

1. Procedure Navigate

Objectives: To navigate in the existing systems. The current path, during the session, is always saved, making it possible to traverse back to the root. This is necessary since the system accepts multiple parents.

```
CODE:
Read button_click from user

Put result into current_id

Append current_id to current_path

SELECT TYPE_ID
FROM ID_REAL
WHERE ID = current_id

Put result into current_type

SELECT IMAGE
FROM SPECIFICATIONS
WHERE TYPE_ID = current_type

Put result into current_image

Retrieve image using current_image

Repeat
      SELECT BUTTONLOC
      FROM PARENT_CHILD
      WHERE PARENT_ID = current_ID

      Put result into button_loc
      Create button using button_loc
end repeat
```

```
Create default buttons
```

2. **Procedure Create_Link**

Objectives: To create a new relation in the system. Normally relations are created by suppliers of system. This procedure is used when attaching different systems and individual components to the ship. Three different, or combination of, cases exist:

1. Create a new individual and a new type. Initially a new image is created. The user is asked to point to the image and name the type. A unique type_id is created and the type information is written to the database. Next step is to query for the location of the button that gives the path to the new child. After creating an unique individual ID, the relations between parent and child, new item and type id, and individual information are written to the database. As a last step the new button is created.

2. Create a new relation, in the database between two existing items. To perform this the user only has to move to the parent item and give a unique id to the child. This is accomplished after the parent is entered in the database.

3. Create a new individual using a existing type. The user provides the button location and the ID of the type that is supposed to be used. This is saved in the database. The following code shows 1 above. 2 and 3 are subsets of 1.

```
CODE:
If necessary create and store new child_image
Ask user for type of link

Create unique type_id     Should be performed by accessing
      the database

Ask user for type_info1

INSERT INTO SPECIFICATIONS
VALUES(type_id, type_info1, child_image, type_info2)

Create new unique child_id
```

```
INSERT INTO ID_REAL
VALUES(child_id,type_id)

INSERT INTO INDIVID
VALUES(child_id,NULL)

Display parent_image and let the user point out the but-
ton_loc

INSERT INTO PARENT_CHILD
VALUES(current_parent, child_id, button_loc)
```

3. **Procedure Delete_Link**

Objectives: To delete a relation in the system. Normally a deletion is performed on the relation between the displayed image and its current parent. This is done when changing the system (i.e. changing inventory). If a parent-child link is deleted there is a possibility that the database could hold items (relations) impossible to reach. A special procedure that checks the database for inconsistencies alleviates this problem.

```
CODE:
DELETE FROM PARENT_CHILD
WHERE PARENT_ID=parent_id AND CHILD_ID=child_id
```

4. **Procedure Attach_Info**

Objectives: To attach individual information to components in the database. This is always performed on the displayed image.

```
CODE:
Retrieve individual information from the user
Put result into text_field
UPDATE INDIVID
SET TEXT_POINTER=text_field
```

```
WHERE ID=current_id
```

5. Procedure Statusreport

Objectives: To show an example of a report. This report counts the number of chairs that do not need repair. The counting of items and similar procedures always assume that inconsistencies do not exist in the database.

```
CODE:
SELECT COUNT(*)
FROM INDIVID I,ID_REAL R, SPECIFICATIONS S
WHERE I.ID=R.ID AND S.TYPE_ID=I.TYPE_ID AND S.NAME='chair'
AND I.STATUS='OK'
```

V. IMPLEMENTATION

This chapter describes a general way of looking at a distributed "non tactical" system onboard a ship. It also gives an example of a process for distributing information in a logistics system.

A. GENERAL

1. Ideas for building a distributed system onboard a ship

As mentioned earlier many of the problems involved in distributing information have not been solved. Despite this, there are several general reasons promoting distribution of information.

Availability. If one part of the ship is damaged or a computer fails it is important that accurate information be available elsewhere. At first glance it could look like it is of little importance to have information about a completely damaged section available, but the information could hide important statistics useful for avoiding problems in the future. To make sure that information be available on more than one site, information has to be duplicated. The updating of information has to be carried out on more than one site with well defined frequencies. Frequency could be defined differently for different kinds of information (i.e. directly, every minute, every hour, every day). The trade-off for doing this is that every update has to be carried out in more than one place using both network and computing resources. If a system is large and widely used, the network capabilities could be the most important part of a distributed system. Redundancy in a distributed system usually implies less response time when asking questions. It is more likely that the requested information is closer to the user so responses are normally faster.

Efficiency. By distributing the information it is possible to customize the system for efficiency. In a fully distributed system there is no single duplication of information. Tables, or parts of tables are located on different sites. It should be possible to carry out

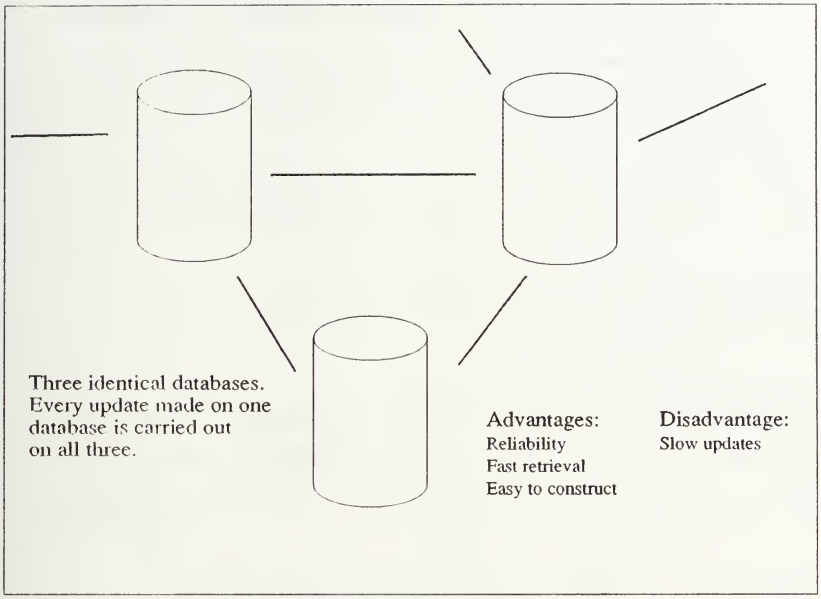


Figure 17: Full redundancy

frequent requests on a users personal system. Less frequently asked questions are allowed to go outside the users own environment. Every update has only to be carried out once. A major drawback with a fully distributed system is that lack of connection to a site that holds necessary information makes the execution of a task impossible. A list of advantages and disadvantages with a distributed system is showed in figure 18.

It is clear, that a redundant and efficient system is a hybrid between a centralized and distributed system, having all or parts of information duplicated at one or more locations.

To determine where and how information will be stored is a tedious job that involves users, technicians and system designers.

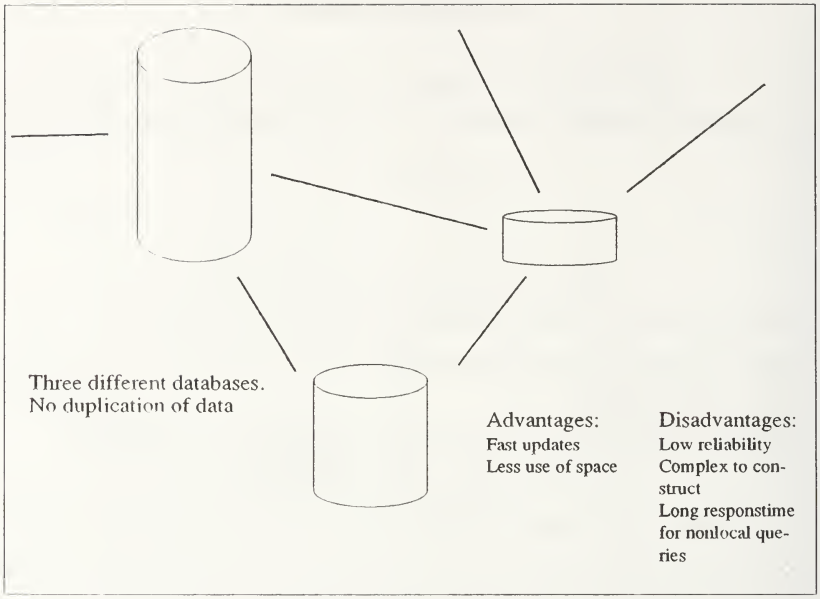


Figure 18: Fully distributed

Design. There are a lot of methods and tools available for designing information systems. Most of them deal with information flows based on needs to execute specific tasks. Specific tasks may include:

- Order an item.
- Move an item.
- Change inventory.
- Deliver an item.

Very few information system developing methods and corresponding tools deal with redundancy and efficiency problems.

B. DISTRIBUTED SYSTEM ONBOARD

The previous section showed clearly that a working system must have duplication of information to increase reliability. It also showed that each duplication that is made must address:

- Efficiency with respect to updates.
- Efficiency with respect to queries.
- Reliability.

The following shows a sample of necessary steps to create a distributed system onboard a ship. Parts of this description include processes that are carried out during normal system development.

1.Start by dividing the ship into functionally autonomous units. Typical examples could be supply, onboard communication, external communications, and weapon systems.

2. Each unit must, independently clearly define what information are necessary to run the unit

3. The next step deals with interaction between units. Information flow between units must be described. This description should include both the content and frequency of flows.

The first three steps are found in almost every software engineering process. The next step directly addresses distributed issues.

Not all information flows have to be implemented as actual flows. In several cases it is more efficient to share a particular storage in the database. So when an update is made the information is not transferred, rather a note is sent to the receiver that an update was made to the database. This note could generate a series of actions at the receiving end.

4. Divide the list of information flows, into two separate lists, one for information that should be transferred and one for information that should be shared.

5. Concentrate on shared information. Construct tables and procedures so that they match all units sharing the information. At this stage it is best to assume that each unit has

it's own database and tables even if they are sharing the same hardware. Shared information could now be stored on a single site or duplicated. Figure 19 shows the logical

EIC_table		
EIC#	NAME	DEP
123456	Bult	1
111222	Nut	1
222333	Screw	1
333444	Washer	1

EIC_table		
EIC#	NAME	DEP
234567	Desk	2
555666	Lamp	2
777888	Raiser	2
999000	Pen	2

Figure 19: Horizontally fragmented tables

fragmentation of a database holding information about EIC items. The left table is owned by Department 1 and the right by Department 2.This is called horizontal fragmentation. Some RDBMS (but certainly not all) can by default handle this fragmentation without concerning the user. In those cases one site (database) is the master database, keeping track of all fragmentations. Only by empiric calculations and extensive testing it can be determined if access and update times are acceptable. If access times are not acceptable, the shared information must be duplicated in both sites. If update times are too slow information should be moved to one site. If both are too slow, the problem must be solved by changing hardware or software or totally rearranging tables and procedures.

6. When data is separated logically it is necessary to deal with physical location. Physical location of data must not necessary coincide with logical location. However, it is preferable to let logical and physical location be the same.

7. Duplication of data for reliability reasons must be dealt with separately. Each logical unit should determine the consequences of lost communication links, hard disk

failure and other run time interferences. Solving this problem involves either backups or more duplication of data.

8. Finally, new calculations and test must take place.

C. DISTRIBUTED SYSTEM WITH RESPECT TO OTHER UNITS

Another approach is necessary when dealing with the interaction between units which are not normally connected. Information can be brought from one unit to another many different ways. This can include using tapes, broadcast, oral, paper, and other not directly digitized communication medias. This may be especially necessary on a ship where radio silence is a must. Here information flow is the key. By clearly defining the information flow between units, including frequencies and volumes, it is possible to build interactive systems. If each unit carries it's own information needs, and information flow is based on possible communications, an 'up to date' information system can be achieved. Figure 20 shows an example of format definitions for information interchange between units.

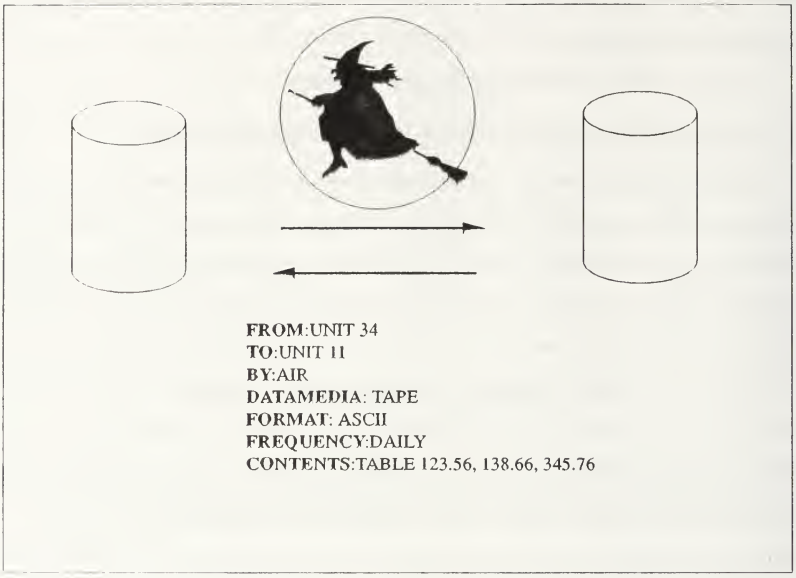


Figure 20: Information flow based on agreements

VI. TEST SYSTEM

This chapter describes the test system which is provided in the appendix. It explains tables and procedures. It also gives a short user manual for the system.

A. REQUIREMENTS FOR THE TEST SYSTEM

The test system is created for the purpose of showing the utilization of the most important parts of the enhanced system.

Separation of data and user interface. To implement the system, even in a test system, it is necessary to show that it is possible to separate the user interface from the stored data. This is necessary not only with respect to stored data about logistic items but also in terms of storing user access paths, creating of unique ID's and other system dependent variables. If as much access as possible is through SQL, user interfaces for additional machines are less complicated to develop.

Use of a well tested commercially available RDBMS. By using a well tested database with network capabilities it is easy to expand the test system from a fairly inexpensive hardware configuration to a fully developed prototype. The selected RDBMS must therefore have working system configurations for at least personal computers (Macintosh, IBM PC), workstations, terminal based mainframes (IBM, VAX) and large mainframes. It is also necessary for the selected RDBM to have a SQL language as close as possible to the ANSI standard. Network capabilities should initially be easier to use than solving complicated distributed queries.

Use of an easy programmable user system tool. To make it possible to easily implement the enhanced system it is preferable to use a tool that it is both commonly used and easy to learn. In the first stage, speed is not the highest priority. It is also important that future work can easily be carried on by other students at NPS.

B. SELECTED TOOLS

The test system is developed using Oracle 1.2 as the RDBMS. The system is implemented on a Macintosh running system 7.0. The access tool for Oracle on the Macintosh is Hypercard. There is also an Oracle C precompiler (ADA and Pascal as well) for Macintosh systems that could be used to create regular Macintosh applications. Oracle 1.2 is created for system 6.x but all tested Oracle calls work fine on system 7.0 after converting the hypercard stacks from 1.2 to 2.1. Some external functions and commands (ResCopy, etc.) used in the stacks will not run on system 7.0 and have to be replaced. The database has to be initialized using the Oracle shell tool to run on system 7.0. It is likely that the test system will run on Oracle 2.0 without conversion.

C. TABLES USED FOR THE TEST SYSTEM

1. Tables

Formats for table columns are listed as they are used by the Oracle RDBMS. Format 'NUMBER' used by ORACLE is the same as format 'NUMERIC' defined in the SQL ANSI standard.

PAR_CHI is used to describe the relation between two items in the system. Each relation is represented by a button on the parent image. The location of the button is described in Hypercard window coordinates. Another way of doing this is to use a percentage of the size of the original image.

PAR_CHI					
PARENT_ID	CHILD_ID	X_LOC	Y_LOC	HEIGHT	WIDTH
NUMBER(10,0)	NUMBER(10,0)	NUMBER(10,0)	NUMBER(10,0)	NUMBER(10,0)	NUMBER(10,0)

CUR_PATH is used to describe the current path for users accessing the system. It could also be used to save different paths used by users or the system.

CUR_PATH		
PARENT_ID	CHILD_ID	USER_ID
NUMBER(10,0)	NUMBER(10,0)	CHAR(10)

SPEC is used to store information about types. Each type has a name and an image related to the type. The image is stored as the full path to the file containing the image. Images are stored in Macintosh 'PICT' format.

SPEC		
TYPE_ID	NAME	IMAGE
NUMBER(10,0)	CHAR(50)	CHAR(70)

ID_REAL is used to relate individual items to their corresponding types.

ID_REAL	
ID	TYPE_ID
NUMBER(10,0)	NUMBER(10,0)

INDIVID is used to store individual information about an item. This table represents tables that could be used to hold history and other interesting individual information about items. This table could also be expanded to include relations to external databases (COSAL, EIC etc.).

INDIVID	
ID	STATUS
NUMBER(10,0)	CHAR(10)

ITEM_ID is used to keep track on the latest issued unique individual ID. This table as implemented never exceeds one row. This solutions will not provide reuse of ID's.

ITEM_ID
NUM
NUMBER(10,0)

TYPE_ID is used exactly in the same manner as ITEM_ID.

TYPE_ID
NUM
NUMBER(10,0)

D. TECHNIQUES USED IN PROCEDURES

All procedures are written on the background script in the TurboArgos 2 stack. The size of the Hypercard stack that holds the program never exceeds 2 cards during a running process. The start size of the stack is slightly less than 200 K. Each time an action involving a traversal of the stored system is performed, the previous card is deleted before a new card is created. All variables except the current card ID are passed between procedures. The current card ID is a global variable.

Log on procedures are performed using the provided template procedures. After starting and logging on to Oracle the first image and corresponding buttons are retrieved. Create new link button, display type info button, etc., are default buttons which are always displayed.

Unique ID's are created by storing an integer value in a table that holds the next available ID. Each time an ID is used, the stored integer value is incremented.

When creating a link to an existing type or item, the user has to use the unique ID to point out the path to the new item.

The Hypercard environment provides a lot of ways to exit the system without using the entrance procedure. To make sure that changes made to the database are always saved, commit is done after each update. This slows down the system. It could be changed to commit after each session, encouraging the user to exit the appropriate way. No inconsistency checks are performed on the database. It is possible to create infinite paths and other illogical information.

E. SHORT USER MANUAL

1. NAVIGATE

To navigate to a displayed item just point and click on it. To go back up to the parent press the up arrow button in the upper left corner.

2. DISPLAY INDIVIDUAL INFORMATION

Depress the magnifier button. This displays individual information, including unique IDs. The individual information can be changed and stored back in the database.

3. DISPLAY TYPE INFORMATION

Depress the type button. Type information including ID is displayed. This information can not be changed.

4. CREATE LINK

When creating a link to a new object and a new type just click on the create link button and follow the instructions.

To create a link to an existing item first go to that item and display the unique ID using the magnifier button. Write down the ID and traverse back to the parent image. Click on the create link button and follow the instructions provided.

To create a link to an existing type first go to an item using that type. Click on the type button and write down the type ID displayed. Go to the parent image and press the create link button and follow the instructions provided.

5. DELETE LINK

Navigate to the child image of the link you want to delete. Press the delete link button. Just the link through the current path is deleted.

VII. FUTURE WORK

This chapter describes suggested future work.

A. GENERAL WORK

It is important to keep in touch with current developments and trends in the DoD. Annual or semi annual meetings with the sponsors at Naval Weapon Station in Concord to check out the current state of SNAP II system development are important. The annual CALS conference must be at least scanned for new adopted standards. Keep up to date versions of new logistic support standards. Scan the DoD for new published standards in the logistic and database area. New information storage tools, including object oriented tools, should be tested and evaluated.

B. MULTIPLE PLATFORMS

The next logical step to prove the usefulness of the enhanced system is to develop a system on multiple platforms. To facilitate this, a new RDBMS tool for the SUN workstations must be installed. An upgraded version of Ingres, or even better, Oracle is a logical choice. This must include tools to provide access from the Macintosh network in the Argos lab in Spanagal 311 to the workstations where the selected RDBMS is installed. The workstation version of the enhanced system could be developed using X-Windows and ADA as the Hypercard equivalent languages.

C. DISTRIBUTED SYSTEMS

Distributed RDBMS systems are not standardized. Future work in distributed systems involves a lot of basic research. To build a distributed test system the first step is to develop a system on multiple platforms. This could be done using two networked Macintosh workstations. Two local RDBMS servers have to be installed, one on each workstation.

APPENDIX

CODE:

```
------------------------------------------------------------------------
-------------------- PROCEDURES FOR ARGOS 2 ----------------------------
------------------------------------------------------------------------

--------------------- NAVIGATE
------------------------------------------------------------------------
on navigate new_parent_id

  put "stefan" into user
  global cur_parent_id

  EXECSQL "SET ERRORINDICATOR :sqlErrNum:"

  --------------------------------- Check if you are going down or up
  EXECSQL "SELECT CHILD_ID FROM CUR_PATH INTO :temp:"&&¬
  "WHERE PARENT_ID = :new_parent_id: and USER_ID = :user:"

  EXECSQL "GET NEXT ROW"
  put the result into sqlerror

  if sqlerror = 0 then
  ------------------------------------Delete path if going up
    EXECSQL "DELETE FROM CUR_PATH"&&¬
    "WHERE PARENT_ID = :new_parent_id: and USER_ID = :user:"
  else
    -------------------------------------Save path if going down
    EXECSQL"INSERT INTO CUR_PATH"&&¬
    " (PARENT_ID,CHILD_ID,USER_ID)"&&¬
    " VALUES(:cur_parent_id:,:new_parent_id:,:user:)"
  end if

  ---------------------------------------------------------------
  put new_parent_id into cur_parent_id
  ------------------------------------------- Retrieve image path
  EXECSQL "SELECT IMAGE "&&¬
  "INTO :ownpath:"&&¬
  "FROM ID_REAL R, SPEC S "&&¬
  "WHERE R.ID = :cur_parent_id: and R.TYPE_ID = S.TYPE_ID"
  ---------------------------------------------------------------
```

```
-------------------------
EXECSQL "GET NEXT ROW" --
-------------------------

domenu New Card
choose select tool
import paint from file ownpath
choose browse tool
------------------------------------------------- Retrieve buttons
EXECSQL "SELECT CHILD_ID,x_loc, y_loc, height, width "&& ¬
"INTO :temp_child_id:,:rightloc:,:downloc:,:height:,:width:"&& ¬
"FROM PAR_CHI"&&¬
"WHERE PARENT_ID = :cur_parent_id:"
-----------------------------------------------------------------

repeat ------------------------------------------ Create buttons
  -------------------------
  EXECSQL "GET NEXT ROW" --
  -------------------------

  put the result into sqlerror
  if sqlerror <> 0 then
    exit repeat
  end if

  domenu New Button
  set the loc of last button to rightloc,downloc
  set the height of last button to height
  set the width of last button to width
  put "on mouseUp" & return &¬
  "doMenu delete card" & return &¬
  "navigate(" & temp_child_id &  ")" &¬
  return & "end mouseUp" &¬
  return into laststring
  set the script of last button to laststring
  set showname of last button to false
  set style of last button to transparent
  set autoHilite of last button to true
end repeat
-------------------------------------------------Retrieve parent
-----------------------------------------------------------------
EXECSQL "SELECT PARENT_ID INTO :parentpath:"&&¬                --
"FROM CUR_PATH"&&¬                                         --
"WHERE USER_ID = 'stefan' AND CHILD_ID = :cur_parent_id:"
```

```
  ------------------------------------------------------------------

  --------------------------
  EXECSQL "GET NEXT ROW"  --
  --------------------------
  put the result into sqlerror

  ------------------If parent exists create parent and delete button
  if (sqlerror = 0 and parentpath > 0) then
    parentbutton(parentpath)
    deletelinkbutton
  end if
  ------------------------------------------------------------------
  EXECSQL "DELETE FROM CUR_PATH"&&¬
  "WHERE USER_ID ='stefan' AND PARENT_ID = :cur_parent_id:"
  ------------------------------------------------------------------
  ----------------------------------------Create default buttons
  createlinkbutton
  gotostartbutton
  infobutton
  typebutton
end navigate

---------------------- CREATELINK
--------------------------------------------------------------------------------
on createlink currentpath
  EXECSQL "SET ERRORINDICATOR :sqlErrNum:"
  answer "Select:" & return & "1. Create link to existing individual."¬
  & return  & "2. Create link using existing type." & return &¬
  "3. Create link to new type." & return  with "1" or "2" or "3"

  put it into choice

  ---------------------------------------------- Get button values
  answer "Please click on the" & return & "upper left corner" &¬
  return & "of the button" with "OK" or "Cancel"

  if it is "OK" then
    ----------------------------------Get upper left corner of new button
    wait until the mouseClick
    put the clickH into upperx
    put the clickV into uppery
    answer "Please click on the" & return & "lower right corner" &¬
    return & "of the button" with "OK" or "Cancel"
```

```
  if it is "OK" then
    ------------------------------Get lower right corner of new button
    wait until the mouseClick
    put the clickH into lowerx
    put the clickV into lowery
  end if
end if

put trunc ((lowerx-upperx)/2+upperx) into locx
put trunc ((lowery-uppery)/2+uppery) into locy
put (lowery-uppery) into height
put (lowerx-upperx) into width

--------------------------------Act accordingly to choices
if choice is "1" then
  ask "Please enter the INDIVIDUAL ID" & return¬
  & "of the item you want to createye a link to"&return
  if it is empty then
    exit createlink
  else
    put it into newpath
    ------------------------------------Store new parent - child link
    EXECSQL "INSERT INTO PAR_CHI"&& ¬
    "VALUES(:currentpath:,:newpath:,:locx:,:locy:,:height:,:width:)"
  end if
end if


if choice is "2" then
  ask "Please enter the TYPE ID" & return¬
  & "of the item you want to createye a link to"&return
  if it is empty then
    exit createlink
  else
    put it into typepath

    put unique_id() into newpath

    ------------------------------------Store new parent - child link
    EXECSQL "INSERT INTO PAR_CHI"&& ¬
    "VALUES(:currentpath:,:newpath:,:locx:,:locy:,:height:,:width:)"

    ------------------------------------Store new individ
```

```
    EXECSQL "INSERT INTO INDIVID"&& ¬
    "VALUES(:newpath:,NULL)"

    ---------------------------------------Store relation to type
    EXECSQL "INSERT INTO ID_REAL"&& ¬
    "VALUES(:newpath:,:typepath:)"
  end if
end if

if choice is "3" then
  answer file "Please select the next picture:" of type PNTG
  put it into path

  if path is not empty then
    put path into pictpath
  else
    exit createlink
  end if

  ask "Please enter a type name"&return
  if it is empty then
    exit createlink
  else
    put it into typename
    put unique_id() into newpath
    put unique_type_id() into typepath

    ---------------------------------------Store new parent - child link
    EXECSQL "INSERT INTO PAR_CHI"&& ¬
    "VALUES(:currentpath:,:newpath:,:locx:,:locy:,:height:,:width:)"

    ---------------------------------------Store new individ
    EXECSQL "INSERT INTO INDIVID"&& ¬
    "VALUES(:newpath:,NULL)"

    ---------------------------------------Store relation to type
    EXECSQL "INSERT INTO ID_REAL"&& ¬
    "VALUES(:newpath:,:typepath:)"

    ---------------------------------------Store type
    EXECSQL "INSERT INTO SPEC"&& ¬
    "VALUES(:typepath:,:typename:,:pictpath:)"

  end if
```

```
  end if
  ------------------------------------------------Create new button
  domenu New Button
  choose browse tool

  set the loc of last button to locx, locy
  set the height of last button to height
  set the width of last button to width
  put numtochar(34) into fnyf
  put "on mouseUp" & return &¬
  "doMenu delete card" & return &¬
  "navigate(" & newpath & ")" &¬
  return & "end mouseUp" &¬
  return into laststring
  set the script of last button to laststring
  set showname of last button to false
  set style of last button to transparent
  set autoHilite of last button to true
  answer "Link created"

end createlink


-------------------- DELETELINK
--------------------------------------------------------------------------
on deletelink currentpath
  answer "This will delete the link between" & return &¬
  "this item and its parent" with "OK" or "Cancel"
  if it is "OK" then
    ---------------------------------------------Retrieve parent
    EXECSQL "SELECT PARENT_ID INTO :parent:"&&¬
    "FROM CUP_PATH"&&¬
    "WHERE CHILD_ID = :currentpath:"
    -------------------------
    EXECSQL "GET NEXT ROW"

    ---------------------------------------------------------------------
    EXECSQL "DELETE FROM PAR_CHI"&&¬
    "WHERE PARENT_ID = :parent: AND CHILD_ID = :currentpath:"

    ------------------------------------------------------Go to the parent
    send mouseUp to button "find parent"
  end if
end deletelink
```

```
---------------------- CREATE UNIQUE ID
--------------------------------------------------------------------------------
function unique_id
  ---------------------------------------------------------------
  EXECSQL "SELECT NUM INTO :number:"&&¬
  "FROM ITEM_ID"

  ---------------------------
  EXECSQL "GET NEXT ROW"

  put number into tempnum
  put number + 1 into number
  ------------------------------------- Store the new highest number
  EXECSQL "UPDATE ITEM_ID" &&¬
  "SET NUM = :number: WHERE NUM = :tempnum:"

  return number
end unique_id

---------------------- CREATE NEW UNIQUE TYPE ID
--------------------------------------------------------------------------------
function unique_type_id
  ---------------------------------------------------------------
  EXECSQL "SELECT NUM INTO :number:"&&¬
  "FROM TYPE_ID"

  ---------------------------
  EXECSQL "GET NEXT ROW"

  put number into tempnum
  put number + 1 into number

  ------------------------------------- Store the new highest number
  EXECSQL "UPDATE TYPE_ID" &&¬
  "SET NUM = :number: WHERE NUM = :tempnum:"

  return number
end unique_type_id



---------------------- RETRIEVE INDIVIDUAL INFORMATION
--------------------------------------------------------------------------------
```

```
on individual item_id
  EXECSQL "SELECT STATUS INTO :status:"&& ¬
  "FROM INDIVID "&&¬
  "WHERE ID = :item_id:"
  ------------------------------------------------------------
  -------------------------
  EXECSQL "GET NEXT ROW"

  ask "Personalized information "&"(ID = "&item_id & ".)"¬
  &return&"Change information if you want" with status

  --------------------------- Change individual information
  execsql "UPDATE INDIVID"&&¬
  "SET STATUS = :it:"&&¬
  "WHERE ID = :item_id:"

end individual

--------------------- RETRIEVE TYPE INFORMATION
-------------------------------------------------------------------------------
on typefetch item_id
  EXECSQL "SELECT NAME,S.TYPE_ID into :name:,:id_type:"&& ¬
  "FROM SPEC S,ID_REAL R "&&¬
  "WHERE R.ID = :item_id: and R.TYPE_ID = S.TYPE_ID "

  -------------------------
  EXECSQL "GET NEXT ROW"

  answer name & return & "(Type_ID = "& id_type & ")" with OK

end typefetch


--------------------- GOTO START
-------------------------------------------------------------------------------
on gotostart
  domenu delete card
  execsql "delete from cur_path"&&¬
  "where user_id = 'stefan' and parent_id > 0"
end gotostart


--------------------- CREATE GO TO START BUTTON
-------------------------------------------------------------------------------
```

```
on gotostartbutton
  domenu New Button
  choose browse tool
  set showname of last button to false
  set the loc of last button to 450,310
  set the height of last button to 35
  set the width of last button to 35
  put numtochar(34) into fnyf
  put "on mouseUp" & return & "gotostart"¬
  & return & "end mouseUp" & return into laststring
  set the script of last button to laststring
  set icon of last button to "fleet return arrow"
  set autoHilite of last button to true
end gotostartbutton


---------- CREATE INFO BUTTON
--------------------------------------------------------------------------
on infobutton
  global cur_parent_id
  domenu New Button
  choose browse tool
  set showname of last button to false
  set the loc of last button to 155,310
  set the height of last button to 35
  set the width of last button to 35
  put numtochar(34) into fnyf
  put "on mouseUp" & return & "" & return &¬
  "individual(" & cur_parent_id &¬
  ")" & return  &¬
  return & "end mouseUp" & return into laststring
  set the script of last button to laststring
  set icon of last button to "closer look"
  set the name of last button to "INFO"
  set autoHilite of last button to true
end infobutton


---------- CREATE GO TO TYPE BUTTON
--------------------------------------------------------------------------
on typebutton
  global cur_parent_id
  domenu New Button
  choose browse tool
```

```
  set showname of last button to false
  set the loc of last button to 210,310
  set the height of last button to 35
  set the width of last button to 35
  put "on mouseUp" & return & "" & return &¬
  "typefetch(" & cur_parent_id &¬
  ")" & return  &¬
  return & "end mouseUp" & return into laststring
  set the script of last button to laststring
  set icon of last button to "type"
  set the name of last button to "TYPE"
  set autoHilite of last button to true
end typebutton


---------- CREATE GOTO PARENT BUTTON
--------------------------------------------------------------------------
on parentbutton parent_path
  domenu New Button
  choose browse tool
  set showname of last button to false
  set the loc of last button to 45,25
  set the height of last button to 45
  set the width of last button to 60
  put "on mouseUp" & return & "domenu delete card" & return &¬
  "navigate(" & parent_path & ")"  &¬
  return & "end mouseUp" & return into laststring
  set the script of last button to laststring
  set icon of last button to find parent
  set the name of last button to "FIND PARENT"
  set autoHilite of last button to true
end parentbutton


---------- CREATE CREATE LINK BUTTON
--------------------------------------------------------------------------
on createlinkbutton
  global cur_parent_id
  domenu New Button
  choose browse tool
  set showname of last button to false
  set the loc of last button to 45,310
  set the height of last button to 35
  set the width of last button to 35
```

```
  put numtochar(34) into fnyf
  put "on mouseUp" & return  & return &¬
  "createlink("& cur_parent_id &¬
  ")" & return  &¬
  return & "end mouseUp" & return into laststring
  set the script of last button to laststring
  set icon of last button to "create link"
  set the name of last button to "CREATE LINK"
  set autoHilite of last button to true
end createlinkbutton


-------------------- CREATE DELETE LINK BUTTON
--------------------------------------------------------------------------------
on deletelinkbutton
  global cur_parent_id
  domenu New Button
  choose browse tool
  set showname of last button to false
  set the loc of last button to 100,310
  set the height of last button to 35
  set the width of last button to 35
  put numtochar(34) into fnyf
  put "on mouseUp" & return  & return &¬
  "deletelink(" & cur_parent_id &¬
  ")" & return  &¬
  return & "end mouseUp" & return into laststring
  set the script of last button to laststring
  set the script of last button to laststring
  set icon of last button to "delete link"
  set the name of last button to "DELETE LINK"
  set autoHilite of last button to true
end deletelinkbutton
```

LIST OF REFERENCES

[GIAN 89] Giannotti B.B., Kevin F. Duffy, Argos: *Design and development of object-oriented, event-driven multimedia database technology in support of the paperless ship.* Thesis NPS Dec 1988, D78372

[MOLI 91] Moliere John P. A., *CALS STANDARDS OVERVIEW,* GTX CORPORATION, 1991

[ORCE 88] ORACLE,*SQL, the quiet revolution* ORCE Systems Software, 1988

[SNAP 86] *SNAP II Supply and Financial Management Subsystem, Volume 1* NAVMASSO DOCUMENT NO. S-1059-005, SB-001, 1986

[TARI 92] Chalousche Tari, *Using Object-Oriented Databases for Implementation of Interactive Electronic Technical Manuals* Thesis NPS March 1992

[ANSI 89] *American National Standard for Information Systems- Database Language - SQL with entegrity Enhancement.* American National Standards Institute, Inc, ANSI X3.135-1989

[MILS 91] *DOD Requirements for a logistic support analysis record (Draft), MIL-STD-1388-2B* Department of Defence 1991.

[SAG 91] *The SQL Access Group, Backgrounder. Overview of the SQL Access Specification* Papers released be the SQL Access Group

BIBLIOGRAPHY

M. Tamer Ozsu, Patrick Valduriez, *Principles of Distributed Database Systems*
Prentice-Hall 1991

Carl Malamud,Van Nostrand, *INGRES Tools for building an Information Architecture*
Reinhold 1989

Mark D. Veljkov, MacLans Local Area Networking with Macintosh,
Scott, Foresman and Company 1988

comp.database
Discussions in UseNet, Jul-Sept 1991

INITIAL DISTRIBUTION LIST

Defense Technical Information Center 2
Cameron Station
Alexandria, VA 221314

Dudley Knox Library 2
Code 0142
Naval Postgraduate School
Monterey, CA 93943

Director of Research Administration 1
Code 012
Naval Postgraduate School
Monterey, CA 93943

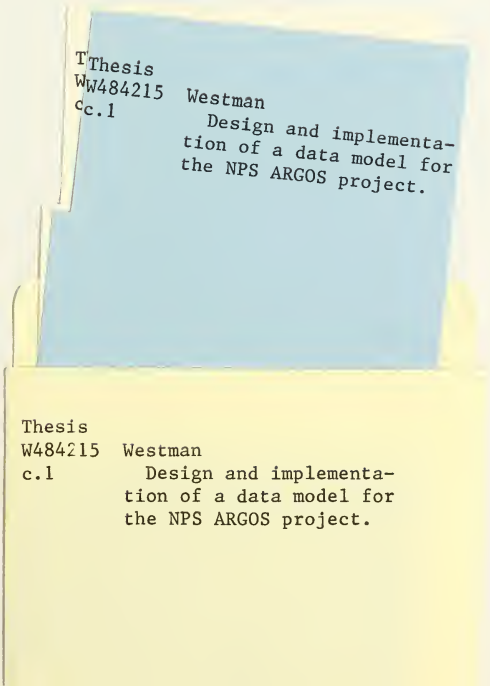
Chairman, Code CS 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

CDR Gino Giannotti 1
NROTC UNIT
RAS 104
University of Texas
Austin, TX 78712-1184

Naval Ocean Systems Center 1
271 Catalina Boulevard
San Diego, CA 92150

Division Head 1
MDS Division
Data Systems Department
Naval Weapon Station
Concord, CA 94520-5000

Phillip B. Stiles 1
Naval Sea Systems Command
Technical Data Division of the Chief Engineer for Logistics Directorate
Washington, D.C. 20362-5101



Thesis
W484215 Westman
c.1 Design and implementation of a data model for the NPS ARGOS project.